EXHIBIT A

EXHIBIT A

XMC Motion Control

XMCSPI Reference

Revision:	Third Draft
Author:	Dave Brown
Date:	Wednesday, Feb 22, 1995
Description:	This document is the XMCAPI reference describing all functions, standard OLE interfaces, and custom OLE interfaces making up the XMCAPI.

Revision History:	4/15/94 (DB)	- First Draft: Initial writing.
	7/1/94 (DB)	- Second Draft: Split design off small business plan, incorporated suggestions.
	2/22/95 (DB)	- Third Draft - Split XMCAPI reference off design guide.

Table of Contents

1.0 Overview

The XMCSPI is the Service Provider Interface (SPI) implemented by every XMC Driver. This layer of software, used by the XMC Motion Component, controls the specific motion control hardware corresponding to the current XMC Driver. Four categories of custom OLE 2.0 interfaces make up the complete XMCSPI layer. Out of the four categories, all XMC Drivers are required to implement one category of interfaces called core interfaces. Implementing the other three interface categories, called extended interfaces, is recommended, but not required to participate in the XMC software model. Implementing all extended interfaces is recommended for it makes the current driver-hardware interaction more precise and more efficient. Since not all hardware will support all extended interfaces, implementing them is not required.

This manual describes all core and extended interfaces making up the complete set of XMCSPI custom OLE interfaces. Chapter 2.0 XMCSPI Interface Categories discusses the purpose of each category and lists the interfaces that fall in each. Chapter 3.0 extends this discussion by describing each interface and the methods they contain.

2.0 XMCSPI Interface Categories

This section describes the categorical grouping of the OLE interfaces found in the XMCSPI. Both standard and custom OLE interfaces are used in the XMCSPI. OLE 2.0 requires the implementation of all interfaces listed in Section *2.1 Standard OLE Interfaces*. XMC requires the implementation of all interfaces listed in section *2.1 Core Interfaces*.

2.1 Standard OLE Interfaces

The following standard OLE interfaces must be supported by each Motion Control Driver. For more information on each interface, see section *2.0 Standard OLE Interfaces* in the XMCAPI Reference manual.

Standard Interfaces

```
IClassFactory
IUnknown
```

2.2 Core Interfaces (Core)

Any absolutely essential functions to motion control problems are in the core set of functions. All motion control XMCSPI drivers must implement these core functions. None of these functions are not allowed to use user-interface objects.

The following custom OLE 2.0 interfaces are in the core set of XMCSPI functions:

Custom Core Interfaces

```
IXMC_DrvCore_AnalogIO
IXMC_DrvCore_DigitalIO
IXMC_DrvCore_DynamicState
IXMC_DrvCore_Encoder
IXMC_DrvCore_EnumInterfaceSupport
IXMC_DrvCore_IO (Abstract)
IXMC_DrvCore_Limits
IXMC_DrvCore_Motion (Dependent)
IXMC_DrvCore_Servo
IXMC_DrvCore_StaticState
IXMC_DrvCore_Stepper
```

NOTE: The IXMC_DrvCore_Servo interface is only required if the motion control hardware controlled by the driver has servo motor support. The same follows for the IXMC_DrvCore_Stepper interface with stepper motors.

Abstract interfaces are only used by other interfaces as an inheritance base. For example, since the **IXMC_DrvCore_DigitalIO** interface inherits from the **IXMC_DrvCore_IO** interface, its implementation contains all methods in both interfaces. See section *2.7.1 Inheritance Relationships* for more information on inheritance relationships between interfaces.

Dependent interfaces are only used after the interfaces they are dependent on have performed certain operations. For example, the **IXMC_DrvCore_Motion** interface is dependent on either the **IXMC_DrvCore_Servo** or the **IXMC_DrvCore_Stepper** interfaces initializing the motion system. See section *2.7.2 Dependency Relationships* for more information on dependency relationships between interfaces.

2.3 Extended Interface Categories

All non-core interfaces in the XMCSPI fall into one of the three extended interface categories. XMC Drivers are not required to implement any of the extended XMCSPI functions. A stub driver, called the XMC Driver Stub, implements all extended interfaces, not supported by the

current driver, that can be emulated using a software algorithm which calls core functions. For more information on how the XMC Driver Stub is used, see *Section 7.0 Driver Administrator, Component, Driver Relationship* in the XMC User Guide manual. Implementing extended interfaces directly in the driver can improve the precision and efficiency of the operation, for the driver implementation has more direct control over the motion control hardware than the XMC Motion Control Component. Even greater precision and performance improvements occur when the driver can direct the motion control hardware to directly perform a procedure defined in one of the extended interfaces.

2.3.1 Extended Interfaces (Ext)

This set of functions consists of all non core functions that do not use user-interface objects such as dialog boxes or message boxes. The following custom OLE 2.0 interfaces are in the set of extended XMCSPI functions:

Custom Ext Interfaces

```
IXMC_DrvExt_Counter
IXMC_DrvExt_Debug (Private)
IXMC_DrvExt_DynamicState
IXMC_DrvExt_Encoder
IXMC_DrvExt_Interrupt
IXMC_DrvExt_IO (Abstract)
IXMC_DrvExt_Joystick
IXMC_DrvExt_Limits
IXMC_DrvExt_Motion (Dependent)
IXMC_DrvExt_StaticState
IXMC_DrvExt_Timer
```

2.3.2 Extended UI Interfaces (ExtUI)

This XMCSPI set consists of all extended XMCSPI functions that require user-interface objects such as dialog boxes or message boxes. For example, when an application requests to tune the servo motors, special input may be needed from the user to complete the process. Because tuning the servo motors is not a required motion control function and may need input from the user, it is in the Extended UI XMCSPI set of functions.

The following custom OLE 2.0 interfaces are in the extended user-interface set of XMCSPI functions:

Custom ExtUI Interfaces

```
IXMC_DrvExtUI_AnalogIO
IXMC_DrvExtUI_Base
IXMC_DrvExtUI_DigitalIO
IXMC_DrvExtUI_DynamicState
IXMC_DrvExtUI_Servo
IXMC_DrvExtUI_StaticState
```

2.3.3 Extended Code Generation Interfaces (ExtCG)

These XMCSPI functions consist of all methods used for code generation only. Most core and extended methods generate code also generate code when the XMC software is run in code-generation mode. But, where core and extended functions may also be used in real-time and mixed modes, code-generation functions cannot. Code-generation functions are only used when running in code-generation mode.

The following custom OLE 2.0 interfaces are in the set of code-generation XMCSPI functions:

Custom ExtCG Interfaces

```
IXMC_DrvExtCG_ProgramFlow
IXMC_DrvExtCG_Operator
IXMC_DrvExtCG_Program
IXMC_DrvExtCG_ProgramMgmt
IXMC_DrvExtCG_Subroutine
IXMC_DrvExtCG_Variable
```

2.4 Vendor Specific Interfaces (ExtVS)

These functions are not part of the set of XMCSPI interfaces. No XMCAPI interface methods call these functions for they are provided by each vendor as a pass-through method giving vendors access to new hardware features not yet supported by the XMC software model. Each implementation of vendor specific custom OLE Interfaces depend on the needs of the hardware vendor. Applications may communicate directly with these interfaces, but in doing so, they automatically become dependent on the specific Motion Control Driver and therefore become hardware-dependent. But, allowing vendor-specific functions to exist gives the software model a flexible way to evolve. For example, if one hardware vendor adds a new feature to their motion control hardware, not provided by any other vendors, they may gain access to the new feature by adding a vendor specific SPI interface to their driver and then directly programming to the SPI interface from within the application. Of course, other drivers will not work with applications that call the hardware dependent SPI interface, for they will not implement the interface. As the feature becomes more useful over time, other hardware vendors will probably add the feature to their hardware. After the feature gains a critical mass of support within the companies supporting the XMC software model, the feature will be added to the XMC software model by becoming a part of both the XMCSPI and XMCAPI. For more information on vendor specific interfaces, see the appropriate XMC Motion Control Extension manual describing the specific XMC Driver which includes a reference for each hardware-dependent driver interface.

2.5 Interface Map

The following table displays all Motion Control Driver, XMCSPI interfaces and the categories they fall in:

Core "IXMCDrvCore_"	Ext "IXMCDrvExt_"	ExtUI "IXMCDrvExtUI_"	ExtCG "IXMCDrvExtCG_"
AnalogIO		AnalogIO Base	AnalogIO*
	Counter Debug (Private)		Counter*
DigitalIO DynamicState	DigitalIO DynamicState	DigitalIO DynamicState	DigitalIO*
Encoder EnumInterface	Encoder		Encoder*
IO (Abstract)	Interrupt IO (Abstract)		Interrupt* IO*
Limits	Joystick Limits		Joystick* Limits*
Motion (Abstract) Servo	Motion (Abstract) Servo	Servo	Motion* Servo*
StaticState Stepper	StaticState	StaticState	Stepper*
	Timer		Timer*

	Operator
	Program
	ProgramFlow
	ProgramMgmt
	Subroutine
	Variable

* These interfaces generate code implicitly. Calling methods, that immediately cause an action on the hardware when run in real-time mode, generate code when run in code-generation mode.

2.6 Interface Relationships

This section describes how closely coupled XMCSPI interfaces relate to one another. There are two ways interfaces relate to one another. Interfaces can inherit from each other and can be dependent on one another. The following sections describe these relationships and the interfaces using them.

2.6.1 Inheritance Relationships

C++ inheritance is used to combine two interfaces in such a way that the derived interface "inherits" all method definitions in the base interface. Only the method definitions within the base interface are inherited. The resulting interface contains implementations of all methods in both the derived and the base interface. The following are the inheritance relationships found in the XMCSPI:

Inheritance Tree

```
IXMC_DrvCore_IO
        |
        |--- IXMC_DrvCore_DigitalIO
        |--- IXMC_DrvCore_AnalogIO

IXMC_DrvExtUI_Base
        |
        |--- IXMC_DrvExtUI_AnalogIO
        |--- IXMC_DrvExtUI_DigitalIO
        |--- IXMC_DrvExtUI_DynamicState
        |--- IXMC_DrvExtUI_Servo
        |--- IXMC_DrvExtUI_StaticState
```

2.6.2 Dependency Relationships

Unlike inheritance relationships, dependency relationships are between two interface implementations. When an interface is dependent on another interface, it needs the interface to perform an operation before it can continue one of its own operations. For example, the **IXMC_DrvCore_Motion** interface is dependent on the IXMC_DrvCore_Stepper interface initialization routine. Before using the **IXMC_DrvCore_Motion** interface, the **IXMC_DrvCore_Stepper** interface must be used to initialize the stepper system. The following are the dependency relationships found in the XMCSPI:

Dependency Tree

```
IXMC_DrvCore_Motion --|          (dependent on)
IXMC_DrvExt_Motion ---|                |
                      |-|              v
                        |--- IXMC_DrvCore_Stepper
                        |--- IXMC_DrvCore_Servo
```

3.0 XMCSPI Reference

3.1 XMC Core Interfaces (Core)

Core functions are the lowest level primitive motion control functions implemented by all hardware vendors that software cannot duplicate in an algorithm that uses other methods, such as those in the Motion Control Driver Stub. This section describes each custom OLE 2.0 interface within the set of core interfaces.

3.1.1 IXMC_DrvCore_AnalogIO Interface

The IXMC_DrvCore_AnalogIO interface inherits from the IXMC_DrvCore_IO interface and implements all of its methods. This interface take care of initializing the analog I/O system used. All I/O work is carried out by the IXMC_DrvCore_IO interface.

Inheritance

```
IXMC_DrvCore_IO
    |
    |------ IXMC_DrvCore_AnalogIO
```

Configuration

```
(*pDrvCore_AnalogIO)->Initialize()
```

3.1.2 IXMC_DrvCore_DigitalIO Interface

Similar to the IXMC_DrvCore_AnalogIO interface, the IXMC_DrvCore_DigitalIO interface is also inherits from the IXMC_DrvCore_IO interface. The IXMC_DrvCore_DigitalIO interface takes care of initializing the digital I/O system, and the IXMC_DrvCore_IO interface takes care of all other operations.

Inheritance

```
IXMC_DrvCore_IO
    |
    |------ IXMC_DrvCore_DigitalIO
```

Configuration

```
(*pDrvCore_DigitalIO)->Initialize()
```

3.1.3 IXMC_DrvCore_DynamicState Interface

The IXMC_DrvCore_DynamicState interface manages the overall state of the motion control system. Included in its management tasks are initializing and shutting down the system. The current state information may be queried or changed at any time after the system is initialized.

State Information

```
typedef struct _XMC_DYNAMICSTATEINFO
{
    XMC_STATICSTATE     *pXMCDC;
    double                 dfActualPosition;
    double                 dfCommandedPosition;
    double                 dfActualVelocity;
    double                 dfCommandedVelocity;
    double                 dfPositionError;
    double                 dfAcceleration;
    double                 dfDeceleration;
    HRESULT                hErrorStatus;
    XMC_OP_MODE            operatingMode;
    XMC_AXIS_DATA         *pLeftLimitData;
    XMC_AXIS_DATA         *pRightLimitData;
} XMC_DYNAMICSTATEINFO;
```

Configuration

```
(*pDrvCore_DynSt)->Reset()
(*pDrvCore_DynSt)->Initialize()
(*pDrvCore_DynSt)->ShutDown()
```

Querying Attributes

```
(*pDrvCore_DynSt)->GetErrorStatus()
(*pDrvCore_DynSt)->GetState()
```

Setting Attributes

```
(*pDrvCore_DynSt)->ClearErrors()
(*pDrvCore_DynSt)->SetState()
```

3.1.4 IXMC_DrvCore_Encoder Interface

The IXMC_DrvCore_Encoder interface implements several optimized functions used to query positions. Calling the *IXMC_DrvCore_DynamicState::GetState()* method will also collect the position information, but at the expense of collecting all other current state data. The methods specified by this interface only collect the data requested.

Querying Attributes

```
(*pDrvCore_Motion)->GetPositionActual()
(*pDrvCore_Motion)->GetPositionError()
```

NOTE: Currently, this interface is classified as a core set of XMCSPI, but it may be changed to an extended set of XMCSPI instead.

3.1.5 IXMC_DrvCore_EnumInterfaceSupport Interface

The IXMC_DrvCore_EnumInterfaceSupport interface is used to query a list of interfaces meeting the specific search criteria set with the *IXMC_DrvCore_EnumInterfaceSupport::SetSearchCriteria()* method..

Inheritance

```
IEnumX
   |
   |--- IXMC_DrvCore_EnumInterfaceSupport
```

Setting Attributes

```
(*pDrvCore_EnumIFace)->SetSearchCriteria()
```

3.1.6 IXMC_DrvCore_IO Interface

This interface consists of all digital input and output related functions. The following methods are available from each Motion Control Driver in the IXMC_DrvCore_DigitalIO interface:

Configuration

```
(*pDrvCore_IO)->Initialize()
```

Actions

```
(*pDrvCore_IO)->Read()
(*pDrvCore_IO)->Write()
```

3.1.7 IXMC_DrvCore_Limits Interface

The IXMC_DrvCore_Limits interface is used to set software limits for a set of axes. When setting limit values, two arrays containing *AXIS_DATA* elements are used to transfer the settings to the method.

Setting Attributes

```
(*pDrvCore_Limits)->SetLimitPositions()
```

NOTE: Currently, this interface is classified as a core set of XMCSPI, but it may be changed to an extended set of XMCSPI instead.

3.1.8 IXMC_DrvCore_Motion Interface

This IXMC_DrvCore_Motion interface is a dependant interface in that it may only be used after either the IXMC_DrvCore_Stepper or IXMC_DrvCore_Servo interfaces are used to initialize the system. All primitive motion control functions that are absolutely necessary to solve motion control software problems are placed in this interface. Also, each function in this set should be a primitive that can not be duplicated through a software algorithm built on top of other primitive functions. The following methods are available from each Motion Control Driver in the IXMC_DrvCore_MotionControl interface:

Querying Attributes

```
(*pDrvCore_Motion)->GetCommandedPosition()
(*pDrvCore_Motion)->GetActualPosition()
(*pDrvCore_Motion)->GetCommandedVelocity()
(*pDrvCore_Motion)->GetActualVelocity()
```

Setting Attributes

```
(*pDrvCore_Motion)->SetVelocity()
(*pDrvCore_Motion)->SetPosition()
```

Actions

```
(*pDrvCore_Motion)->Kill()
(*pDrvCore_Motion)->MoveAbs()
(*pDrvCore_Motion)->MoveContinous()
(*pDrvCore_Motion)->Stop()
```

3.1.9 IXMC_DrvCore_Servo Interface

The IXMC_DrvCore_Servo interface handles all operations specific to servo motors. All motion operations are performed by the IXMC_DrvCore_Motion interface. The following methods are in the IXMC_DrvCore_Servo interface:

Dependency

```
IXMC_DrvCore_Motion
    |
    |----- IXMC_DrvCore_Servo
```

Configuration

```
(*pDrvCore_Servo)->Initialize()
(*pDrvCore_Servo)->Tune()
```

3.1.10 IXMC_DrvCore_StaticState Interface

The IXMC_DrvCore_StaticState interface manages all static state information defining the system. Like the current state information, the device context information may be queried at any time after the system is initialized. But, unlike the current state information, the device context information is only set during the initial setup of the system.

State Information

```
typedef struct _XMC_STATICSTATEINFO
{
    double          dfEncoderCountsPerRevolution;
    double          dfMMPerRevolution;
    int         cbAxisCount;
    int         cbAxisMax;
    XMCMOTORTYPE    motorType;
} XMC_STATICSTATEINFO;
```

Querying Attributes

```
(*pDrvCore_StatSt)->GetState()
```

Setting Attributes

```
(*pDrvCore_StatSt)->SetState()
```

3.1.11 IXMC_DrvCore_Stepper Interface

The IXMC_DrvCore_Stepper interface handles all operations specific to stepper motors. All motion operations are performed by the IXMC_DrvCore_Motion interface. The following methods are in the IXMC_DrvCore_Stepper interface:

Dependency

```
IXMC_DrvCore_Motion
    |
    |----- IXMC_DrvCore_Stepper
```

Configuration

```
(*pDrvCore_Stepper)->Initialize()
```

3.2 XMC Extended Interfaces (Ext)

Extended interfaces contain methods that are not required. These interfaces are either duplicated by the Motion Control Driver Stub or contain functionality not essential in most motion control problems. However, implementing these interfaces will optimize the performance of the motion control system for a particular set of hardware. This section describes all extended interfaces within XMCSPI.

3.2.1 IXMC_DrvExt_Counter Interface

The IXMC_DrvExt_Counter interface is used to operate a counter implemented by the motion control hardware.

Querying Attributes

```
(*pDrvExt_Counter)->GetCounter()
```

Actions

```
(*pDrvExt_Counter)->EnableInterrupt()
(*pDrvExt_Counter)->Reset()
(*pDrvExt_Counter)->Start()
(*pDrvExt_Counter)->Stop()
```

3.2.2 IXMC_DrvExt_Debug Interface (Private)

The **IXMC_DrvExt_Debug** interface is a private interface used by the Motion Control Component, if available. When implemented, the interface is used when debugging the Motion Control Driver.

Setting Attributes

```
(*pDrvExt_Debug)->SetRawDataInDumpFile()
(*pDrvExt_Debug)->SetRawDataOutDumpFile()
(*pDrvExt_Debug)->SetXMCSPILogFile()
```

Actions

```
(*pDrvExt_Debug)->EnableRawDataDump()
(*pDrvExt_Debug)->EnableXMCSPILogging()
```

3.2.3 IXMC_DrvExt_DynamicState Interface

The **IXMC_DrvExt_DynamicState** interface manages the extended, dynamic state of the motion control system. The extended current state information may be queried or changed at any time after the system is initialized.

State Information

```
typedef struct _XMC_EXTDYNAMICSTATEINFO
{
    DWORD                          dwSupportedPoperties;
    XMC_EXTSTATICSTATEINFO        *pExtXMCDC;
    double                         dfLimitDeceleration;
    double                         dfJogVelocityHigh;
    double                         dfJogVelocityLow;
    double                         dfJoystickVelocityHigh;
    double                         dfJoystickVelocityLow;
    double                         dfFeedRate;
    POS_DATA                      *pHomePosition;
    POS_DATA                      *pZeroPosition;
    POS_DATA                      *pLimitSWFwdPosition;
    POS_DATA                      *pLimitSWBwdPosition;
    double                         dfLimitDeceleration;
    BOOL                           bJoystick;
    BOOL                           bAxisScaling;
    BOOL                           bPathScaling;
    BOOL                           bDataCapture;
    BOOL                           bFeedRate;
    BOOL                           bInterpolation;
    BOOL                           bLimitSW;
    BOOL                           bLimitHW;
    BOOL                           bIODataCapture;
} XMC_EXTDYNAMICSTATEINFO;
```

Querying Attributes

```
(*pDrvCore_DynSt)->GetState()
```

Setting Attributes

```
(*pDrvCore_DynSt)->SetState()
```

3.2.4 IXMC_DrvExt_DigitalIO Interface

The IXMC_DrvExt_DigitalIO interface handles all extended IO functionaly.

Actions

```
(*pDrvExt_DigitalIO)->EnableInputFunctions()
(*pDrvExt_DigitalIO)->EnableOutputFunctions()
```

Setting Attributes

```
(*pDrvExt_DigitalIO)->SetInputFunction()
(*pDrvExt_DigitalIO)->SetInputDebounceTime()
(*pDrvExt_DigitalIO)->SetInputActiveLevel()
(*pDrvExt_DigitalIO)->SetOutputFunction()
(*pDrvExt_DigitalIO)->SetOutputActiveLevel()
(*pDrvExt_DigitalIO)->SetOutputStates()
```

3.2.5 IXMC_DrvExt_Encoder Interface

The IXMC_DrvExt_Encoder interface is used to manipulate encoders in the motion control system.

Setting Attributes

```
(*pDrvExt_Encoder)->SetOutputAbsPosition()
(*pDrvExt_Encoder)->SetOutputRelPosition()
```

3.2.6 IXMC_DrvExt_Interrupt Interface

The IXMC_DrvExt_Interrupt interface is used to set, get, and enable interrupt handlers

Querying Attributes

```
(*pDrvExt_Int)->GetHandler()
```

Setting Attributes

```
(*pDrvExt_Int)->SetHandler()
```

Actions

```
(*pDrvExt_Int)->Enable()
```

3.2.7 IXMC_DrvExt_IO Interface

This interface consists of all digital input and output related functions. The following methods are available from each Motion Control Driver in the IXMC_DrvExt_DigitalIO interface:

Actions

```
(*pDrvExt_IO)->EnableDataCapture();
```

3.2.8 IXMC_DrvExt_Joystick Interface

The IXMC_DrvExt_Joystick interface is used to enable and disable a hardware joystick. When enabled, all other motion operations are disabled. The following methods are in this interface:

Setting Attributes

```
(*pDrvExt_Joystick)->SetVelocityHigh()
(*pDrvExt_Joystick)->SetVelocityLow()
(*pDrvExt_Joystick)->SetupElectronics()
```

Actions

```
(*pDrvExt_Joystick)->Enable()
```

3.2.9 IXMC_DrvExt_Limits Interface

The IXMC_DrvExt_Limits interface is used to manipulate both hardware and software limits. The following methods are in this interface:

Querying Attributes

```
(*pDrvExt_Limits)->GetSoftwareLimitPositions()
```

Setting Attributes

```
(*pDrvExt_Limits)->SetSoftwareLimitPositions()
(*pDrvExt_Limits)->SetDeceleration()
```

Actions

```
(*pDrvExt_Limits)->EnableHardwareLimits()
(*pDrvExt_Limits)->EnableSoftwareLimits()
```

3.2.10 IXMC_DrvExt_Motion Interface

This interface consists of extra motion control functions that may or may not be implemented by the motion control hardware. If a hardware implementation is unavailable for a particular function, the Motion Component calls the Motion Control Driver Stub, which implements the functionality in software. The following methods are available from either the Motion Control Driver or the Motion Control Driver Stub in the IXMC_DrvExt_MotionControl interface:

Querying Attributes

```
(*pDrvExt_Motion)->GetFeedRate()
(*pDrvExt_Motion)->GetAxisScaling()
(*pDrvExt_Motion)->GetPathScaling()
(*pDrvExt_Motion)->GetMaxAcceleration()
(*pDrvExt_Motion)->GetMaxDeceleration()
(*pDrvExt_Motion)->GetMaxVelocity()
(*pDrvExt_Motion)->GetHomePosition()
(*pDrvExt_Motion)->IsDataCaptureOn()
(*pDrvExt_Motion)->IsFeedRateOn()
(*pDrvExt_Motion)->IsAxisScalingOn()
(*pDrvExt_Motion)->IsPathScalingOn()
(*pDrvExt_Motion)->IsInterpolationOn()
```

Setting Attributes

```
(*pDrvExt_Motion)->SetJogVelocityHigh()
(*pDrvExt_Motion)->SetJogVelocityLow()
(*pDrvExt_Motion)->S tFeedRate()
(*pDrvExt_Motion)->SetAxisScaling()
(*pDrvExt_Motion)->SetPathScaling()
(*pDrvExt_Motion)->SetMaxAcceleration()
(*pDrvExt_Motion)->SetMaxD celeration()
(*pDrvExt_Moti n)->SetMaxVelocity()
(*pDrvExt_Motion)->SetZeroPosition()
(*pDrvExt_Motion)->SetHomePosition()
```

Action

```
(*pDrvExt_M tion)->EnableFeedRate()
(*pDrvExt_M tion)->Enabl Interpolation()
(*pDrvExt_Motion)->EnableAxisScaling()
(*pDrvExt_Motion)->EnablePathScaling()
(*pDrvExt_Motion)->GoHome()
(*pDrvExt_Motion)->GoZero()
(*pDrvExt_Motion)->MoveRel()
```

Geometric Moves

```
(*pDrvExt_Motion)->Arc()
(*pDrvExt_Motion)->Path()
```

3.2.11 IXMC_DrvExt_StaticState Interface

The **IXMC_DrvExt_StaticState** interface manages all extended static state information defining the system. Like the current state information, the device context information may be queried at any time after the system is initialized. But, unlike the current state information, the device context information is only set during the initial setup of the system.

State Information

```
typedef struct _XMC_EXTSTATICSTATEINFO                    —
{
   DWORD                              dwSupportedProperties;
   double                             dfAxisScalingFactor;
   double                             dfPathScalingFactor;
   double                             dfMaxAcceleration;
   double                             dfMaxDeceleration;
   double                             dfMaxVelocity;
} XMC_EXTSTATICSTATEINFO;
```

Querying Attributes

```
(*pDrvCore_DC)->GetState()
```

Setting Attributes

```
(*pDrvCore_DC)->SetState()
```

3.2.12 IXMC_DrvExt_Timer Interface

The **IXMC_DrvExt_Timer** interface is used to manipulate a timer on the motion control hardware.

Querying Attributes

```
(*pDrvExt_Timer)->GetTime()
(*pDrvExt_Timer)->GetDelay()
(*pDrvExt_Timer)->GetResolution()
```

Setting Attributes

```
(*pDrvExt_Timer)->SetDelay()
```

Actions

```
(*pDrvExt_Timer)->DoDelay()
(*pDrvExt_Timer)->EnableInterrupt()
(*pDrvExt_Timer)->Reset()
(*pDrvExt_Timer)->Start()
(*pDrvExt_Timer)->Stop()
(*pDrvExt_Timer)->TriggerInterrupt()
```

3.3 XMC Extended UI Interfaces (ExtUI)

All extended UI custom OLE 2.0 interfaces use user-interface resources such as dialog boxes or windows to interact with the user. Some interfaces may gather information from the user, where others may just display information.

3.3.1 IXMC_DrvExtUI_AnalogIO Interface

The IXMC_DrvExtUI_AnalogIO interface allows the user to interactively view and set the parameters used when initializing the analog IO system.

Inheritance

```
IXMC_DrvCore_IO
    |
    |------ IXMC_DrvCore_DigitalIO
```

3.3.2 IXMC_DrvExtUI_AnalogIO Interface

The IXMC_DrvExtUI_AnalogIO interface allows the user to interactively view and set the parameters used when initializing the analog IO system.

Inheritance

```
IXMC_DrvExtUI_Base
    |
    |------ IXMC_DrvExtUI_AnalogIO
```

3.3.3 IXMC_DrvExtUI_Base Interface

The IXMC_DrvExtUI_Base interface is an abstract base derived off by all other Extened UI interfaces. The two methods defined in this interface, implemented by the deriving interface, are used to initialize and view data.

Configuration

```
(*pDrvExtUI_Base)->Initialize()
(*pDrvExtUI_Base)->View()
```

3.3.4 IXMC_DrvExtUI_DynamicState Interface

The IXMC_DrvExtUI_DynamicState interface allows the user to interactively view and set all dynamic parameters used in the motion control system.

Inheritance

```
IXMC_DrvExtUI_Base
    |
    |------ IXMC_DrvExtUI_DynamicState
```

3.3.5 IXMC_DrvExtUI_StaticState Interface

The IXMC_DrvExtUI_StaticState interface allows the user to interactively view and set all static system parameters used when initializing the motion control system.

Inheritance

```
IXMC_DrvExtUI_Base
    |
    |------ IXMC_DrvExtUI_StaticState
```

3.3.6 IXMC_DrvExtUI_DigitalIO Interface

The IXMC_DrvExtUI_DigitalIO interface allows the user to interactively view and set the parameters used when initializing the digital IO system.

Inheritance

```
IXMC_DrvExtUI_Base
    |
    |------ IXMC_DrvExtUI_DigitalIO
```

3.3.7 IXMC_DrvExtUI_Servo Interface

The IXMC_DrvExtUI_Servo interface allows the user to interactively view and set the servo system parameters. Also, the interface allows the user to interactively tune the PID settings used.

Inheritance

```
IXMC_DrvExtUI_Base
    |
    |------ IXMC_DrvExtUI_Servo
```

Configuration

```
(*pDrvExt_Motion)->Tune()
```

3.4 XMC Extended Code Generation Interfaces (ExtCG)

Code generation interfaces are used by the Motion Control Component to generate the appropriate vendor-specific control-codes used to create a motion control program. Unlike other interface supported by each Motion Control Driver, the code-generation interfaces are never used in real-time mode.

NOTE: Currently, the implementations of all code generation interfaces are a direct mapping of the Motion Control Component code generation interfaces and are under investigation.

3.4.1 IXMC_DrvExtCG_Operator Interface

The IXMC_DrvExtCG_Operator interface is used to generate general program code such as boolean, logical, simple conditional, and mathematical operators. The following methods are in the interface:

Bitwise

```
(*pDrvExtCG_Op)->BitAnd()
(*pDrvExtCG_Op)->BitNot()
(*pDrvExtCG_Op)->BitOr()
(*pDrvExtCG_Op)->BitShiftL()
(*pDrvExtCG_Op)->BitShiftR()
(*pDrvExtCG_Op)->BitXOr()
```

Logical

```
(*pDrvExtCG_Op)->And()
(*pDrvExtCG_Op)->Not()
(*pDrvExtCG_Op)->Or()
```

Mathematical

```
(*pDrvExtCG_Op)->Add()
(*pDrvExtCG_Op)->Bit()
(*pDrvExtCG_Op)->Div()
(*pDrvExtCG_Op)->LeftParen()
(*pDrvExtCG_Op)->Mult()
(*pDrvExtCG_Op)->RightParen()
(*pDrvExtCG_Op)->Sqrt()
(*pDrvExtCG_Op)->Sub()
```

Trigonometric

```
(*pDrvExtCG_Op)->ACos()
(*pDrvExtCG_Op)->ASin()
(*pDrvExtCG_Op)->ATan()
(*pDrvExtCG_Op)->Cos()
(*pDrvExtCG_Op)->Sin()
(*pDrvExtCG_Op)->Tan()
```

Labels/Strings/Comments

```
(*pDrvExtCG_Op)->ASCIIChar()
(*pDrvExtCG_Op)->Comment()
(*pDrvExtCG_Op)->DeclarLabel()
(*pDrvExtCG_Op)->String()
```

Conditional

```
(*pDrvExtCG_Op)->Equal()
(*pDrvExtCG_Op)->GreaterThan()
(*pDrvExtCG_Op)->GreaterThanEqual()
(*pDrvExtCG_Op)->LessThan()
(*pDrvExtCG_Op)->LessThanEqual()
(*pDrvExtCG_Op)->NotEqual()
```

3.4.2 IXMC_DrvExtCG_Program Interface

The IXMC_DrvExtCG_Program interface is used to generate a program shell using the appropriate vendor-specific control-codes. The following methods are in the interface:

Programmming

```
(*pDrvExtCG_Prog)->Define()
(*pDrvExtCG_Prog)->End()
(*pDrvExtCG_Prog)->Exit()
(*pDrvExtCG_Prog)->Sleep()
```

3.4.3 IXMC_DrvExtCG_ProgramFlow Interface

The IXMC_DrvExtCG_ProgramFlow interface is used to generate program flow code. Loops and if/then statements are several examples of contitional code generated. The following methods are in the interface:

If/Else

```
(*pDrvExtCG_ProgFlow)->IfOpen()
(*pDrvExtCG_ProgFlow)->IfClose()
(*pDrvExtCG_ProgFlow)->Else()
(*pDrvExtCG_ProgFlow)->EndIf()
```

While

```
(*pDrvExtCG_ProgFlow)->WhileOpen()
(*pDrvExtCG_ProgFlow)->WhileClose()
(*pDrvExtCG_ProgFlow)->EndWhile()
```

Repeat

```
(*pDrvExtCG_ProgFlow)->Repeat()
(*pDrvExtCG_ProgFlow)->UntilOpen()
(*pDrvExtCG_ProgFlow)->UntilClose()
```

Loops

```
(*pDrvExtCG_ProgFlow)->Loop()
(*pDrvExtCG_ProgFlow)->EndLoop()
```

Misc.

```
(*PDrvExtCG_ProgFlow)->Break()
(*PDrvExtCG_ProgFlow)->GoTo()
```

3.4.4 IXMC_DrvExtCG_ProgramMgmt Interface

The IXMC_DrvExtCG_ProgramFlow interface is used to generate program flow code. Loops and if/then statements are several examples of contitional code generated. The following methods are in the interface:

Programming

```
(*pDrvExtCG_ProgMgmt)->Run()
(*pDrvExtCG_ProgMgmt)->Stop()
```

Output

```
(*pDrvExtCG_ProgMgmt)->EnableOutput()
(*pDrvExtCG_ProgMgmt)->SetOutput()
```

3.4.5 IXMC_DrvExtCG_Subroutine Interface

The IXMC_DrvExtCG_Subroutine interface is used to generate subroutines and calls to subroutines. The following methods are in the IXMC_DrvExtCG_Subroutine interface:

General

```
(*pDrvExtCG_Sub)->Define()
(*pDrvExtCG_Sub)->Call()
(*pDrvExtCG_Sub)->End()
```

3.4.6 IXMC_DrvExtCG_Variable Interface

The IXMC_DrvExtCG_Variable interface is used to define and assign values to variables. The following methods are included in the interface:

Definition

```
(*pDrvExtCG_Var)->Define()
```

Assignment

```
(*pDrvExtCG_Var)->AssignNumeric()
(*pDrvExtCG_Var)->AssignBinary()
(*pDrvExtCG_Var)->AssignString()
```

Appendix A

A.1 Reviewer Feedback

All feedback is important in order to evolve the specification into a standard software model for motion control hardware. Please, at a minimum, respond to the following issues:

1. Which areas, ideas, or concepts within the specification seem confusing to you?
2. Are there any main features missing from the specification that you would find useful?
3. Are there any specific scenario-maps you would like to see in the next revision?
4. Are there any specific C or C++ code examples you would like to see?
5. Are there any specific Visual Basic code examples you would like to see?

A.1.1 Correspondence

If possible, please send all feedback over email to the following address:

Dave Brown
ROY-G-BIV Corporation
Compuserve: 72103,2235
Internet: 72103.2235@compuserve.com

Otherwise, if you do not have access to email, please mail all correspondence to the following address:

Dave Brown
ROY-G-BIV Corporation
P.O. Box 1278
White Salmon, WA. 98672